UNLOCK TABLES;

CREATE DATABASE CRYPTOPUNK;

USE CRYPTOPUNK;

SELECT \* FROM PRICEDATA;

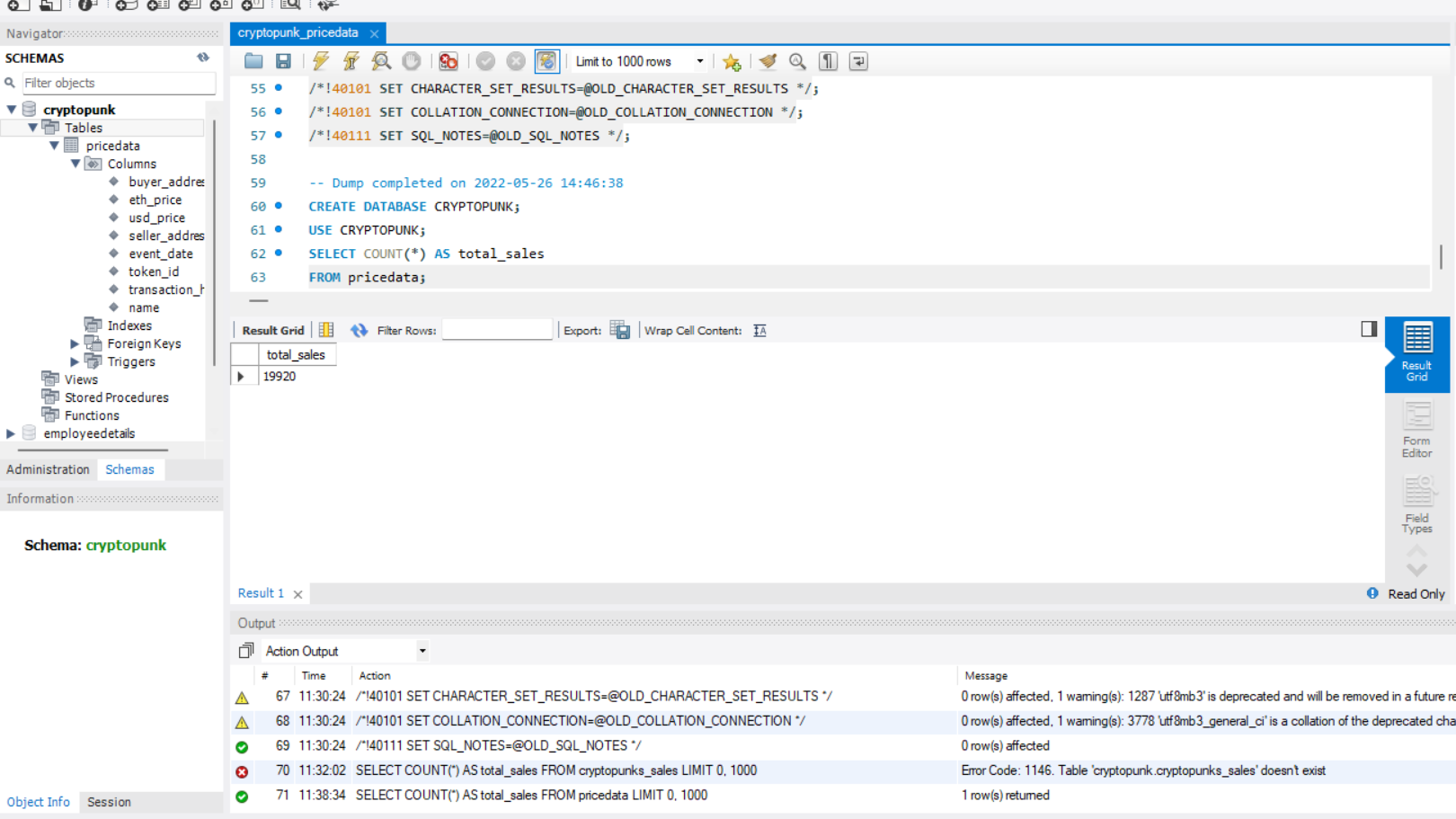
1. \*\*How many sales occurred during this time period?\*\*

```sql

SELECT COUNT(\*) AS total\_sales

FROM pricedata;

```



2. \*\*Return the top 5 most expensive transactions (by USD price) for this data set. Return the name, ETH price, and USD price, as well as the date.\*\*

```sql

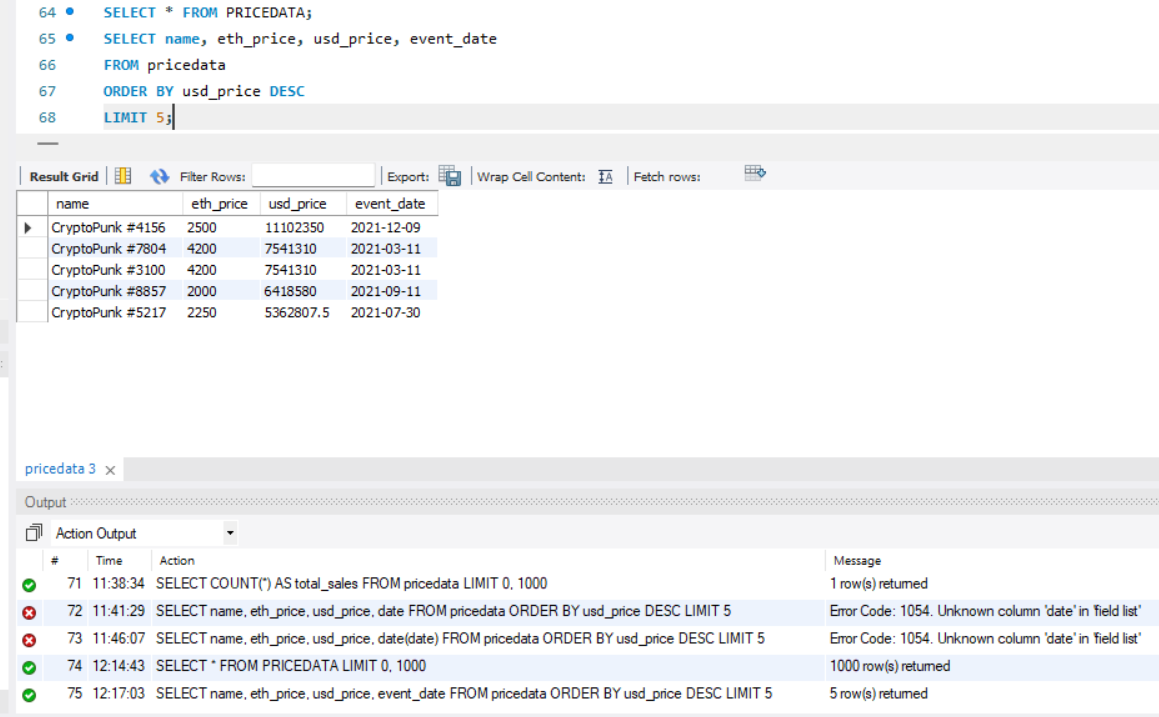
SELECT name, eth\_price, usd\_price, event\_date

FROM pricedata

ORDER BY usd\_price DESC

LIMIT 5;

```



3. \*\*Return a table with a row for each transaction with an event column, a USD price column, and a moving average of USD price that averages the last 50 transactions.\*\*

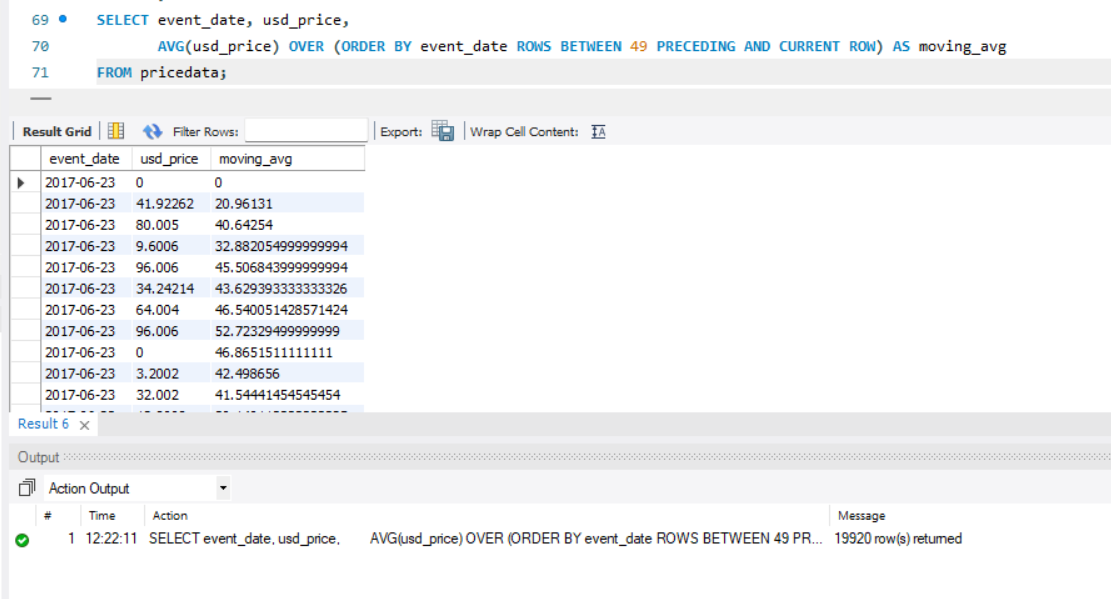
```sql

SELECT event\_date, usd\_price,

AVG(usd\_price) OVER (ORDER BY event\_date ROWS BETWEEN 49 PRECEDING AND CURRENT ROW) AS moving\_avg

FROM pricedata;

```



4. \*\*Return all the NFT names and their average sale price in USD. Sort descending. Name the average column as average\_price.\*\*

```sql

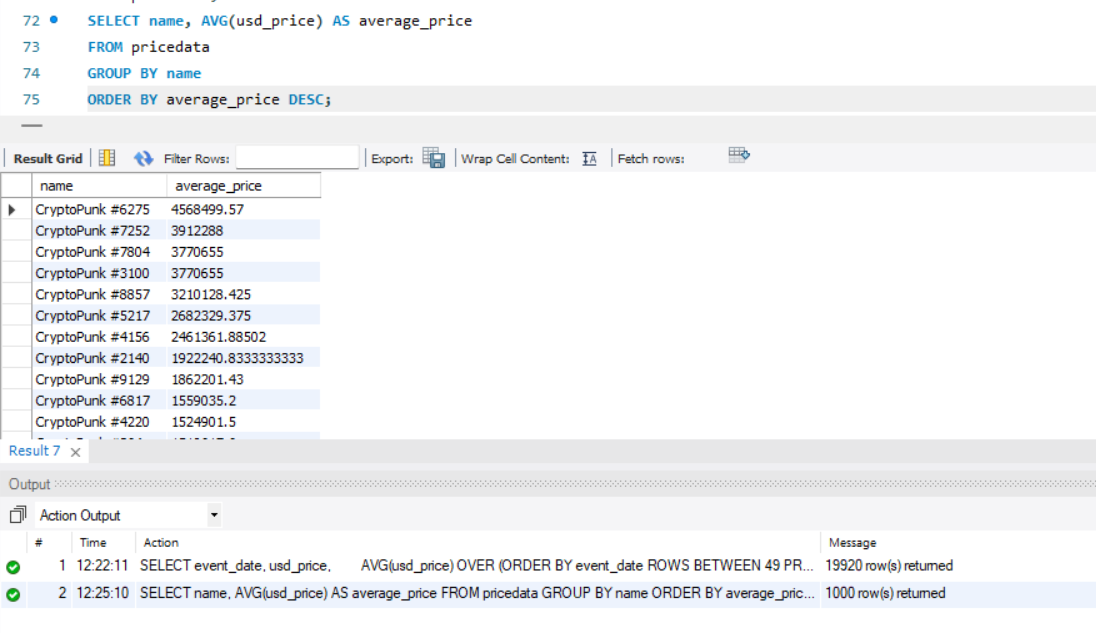
SELECT name, AVG(usd\_price) AS average\_price

FROM pricedata

GROUP BY name

ORDER BY average\_price DESC;

```



5. \*\*Return each day of the week and the number of sales that occurred on that day of the week, as well as the average price in ETH. Order by the count of transactions in ascending order.\*\*

```sql

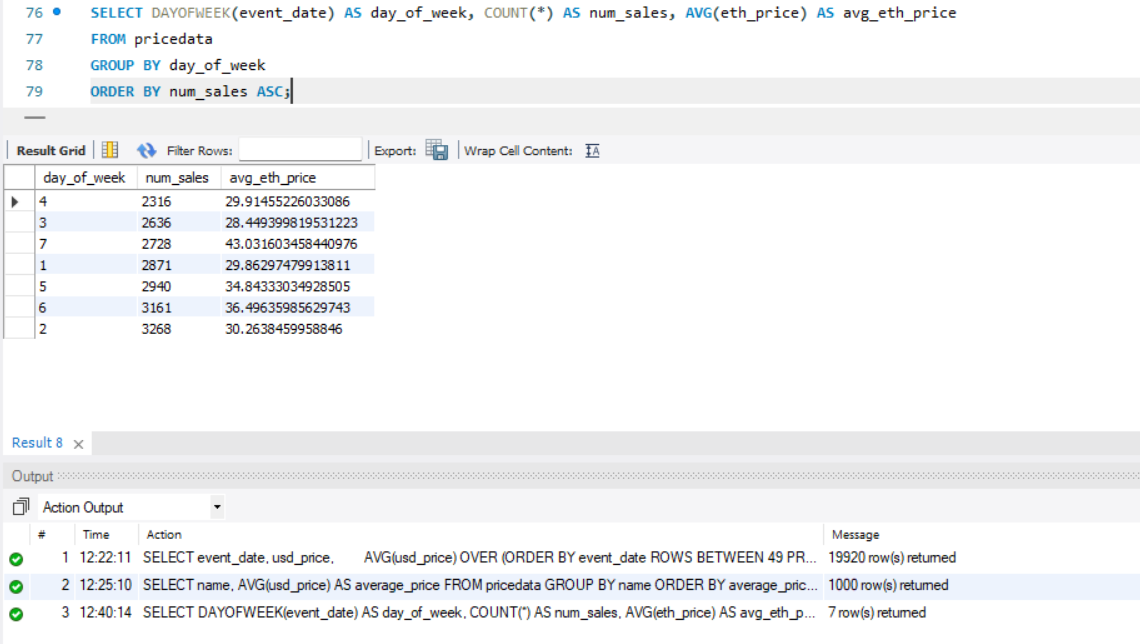
SELECT DAYOFWEEK(event\_date) AS day\_of\_week, COUNT(\*) AS num\_sales, AVG(eth\_price) AS avg\_eth\_price

FROM pricedata

GROUP BY day\_of\_week

ORDER BY num\_sales ASC;

```



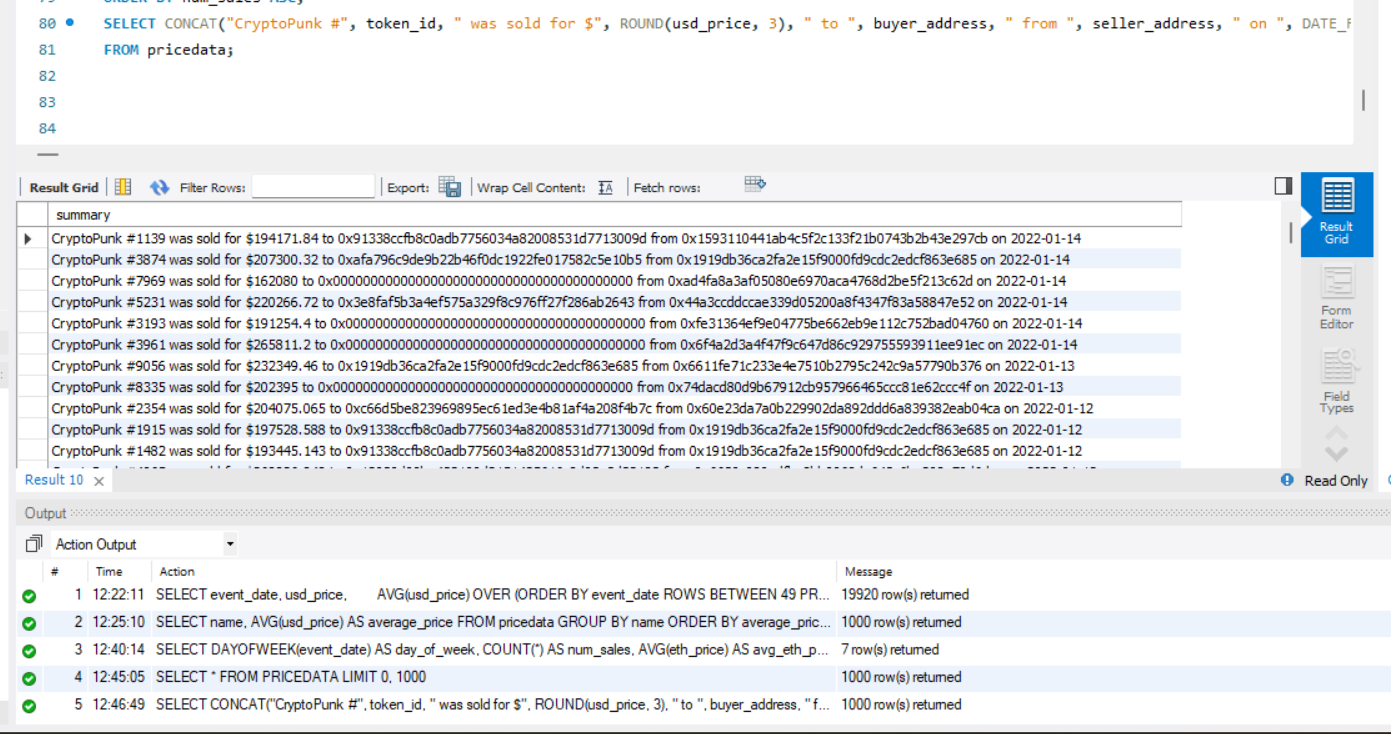
6. \*\*Construct a column that describes each sale and is called summary.\*\*

```sql

SELECT CONCAT("CryptoPunk #", token\_id, " was sold for $", ROUND(usd\_price, 3), " to ", buyer\_address, " from ", seller\_address, " on ", DATE\_FORMAT(event\_date, '%Y-%m-%d')) AS summary

FROM pricedata;

```



7. \*\*Create a view called “1919\_purchases” and contains any sales where “0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685” was the buyer.\*\*

```sql

CREATE VIEW 1919\_purchases AS

SELECT \*

FROM pricedata

WHERE buyer\_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';

```



8. \*\*Create a histogram of ETH price ranges. Round to the nearest hundred value.\*\*

```sql

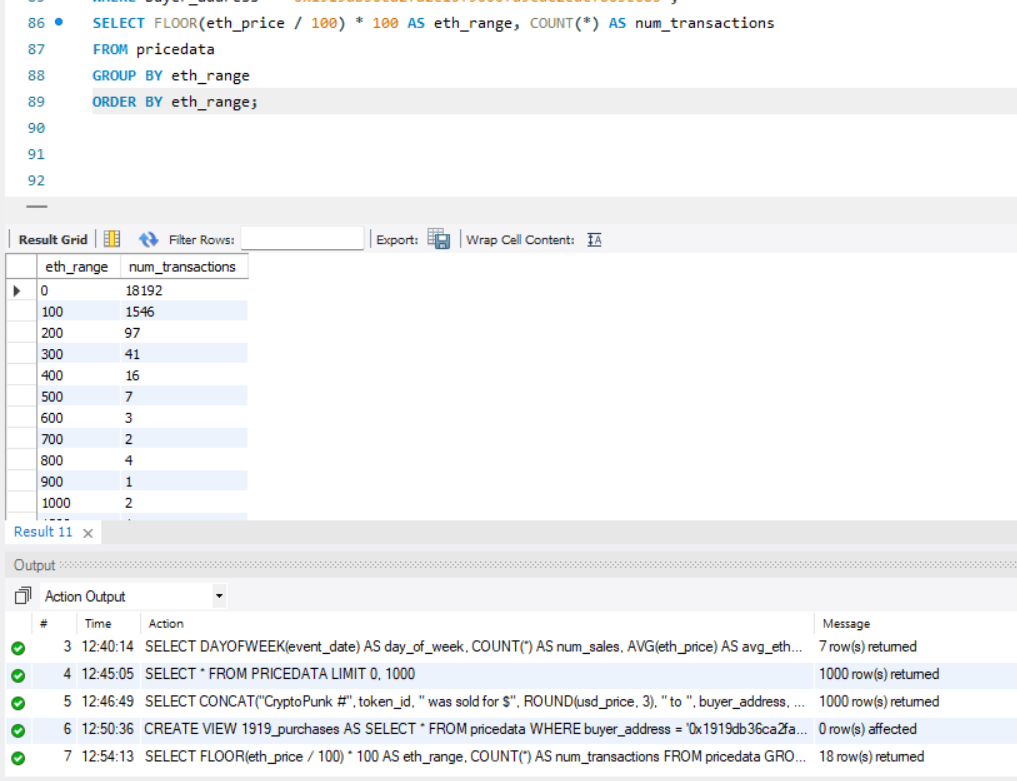
SELECT FLOOR(eth\_price / 100) \* 100 AS eth\_range, COUNT(\*) AS num\_transactions

FROM pricedata

GROUP BY eth\_range

ORDER BY eth\_range;

```



9. \*\*Return a unioned query that contains the highest price each NFT was bought for and a new column called status saying “highest” with a query that has the lowest price each NFT was bought for and the status column saying “lowest”.\*\*

```sql

(SELECT name, MAX(usd\_price) AS price, 'highest' AS status

FROM pricedata

GROUP BY name)

UNION

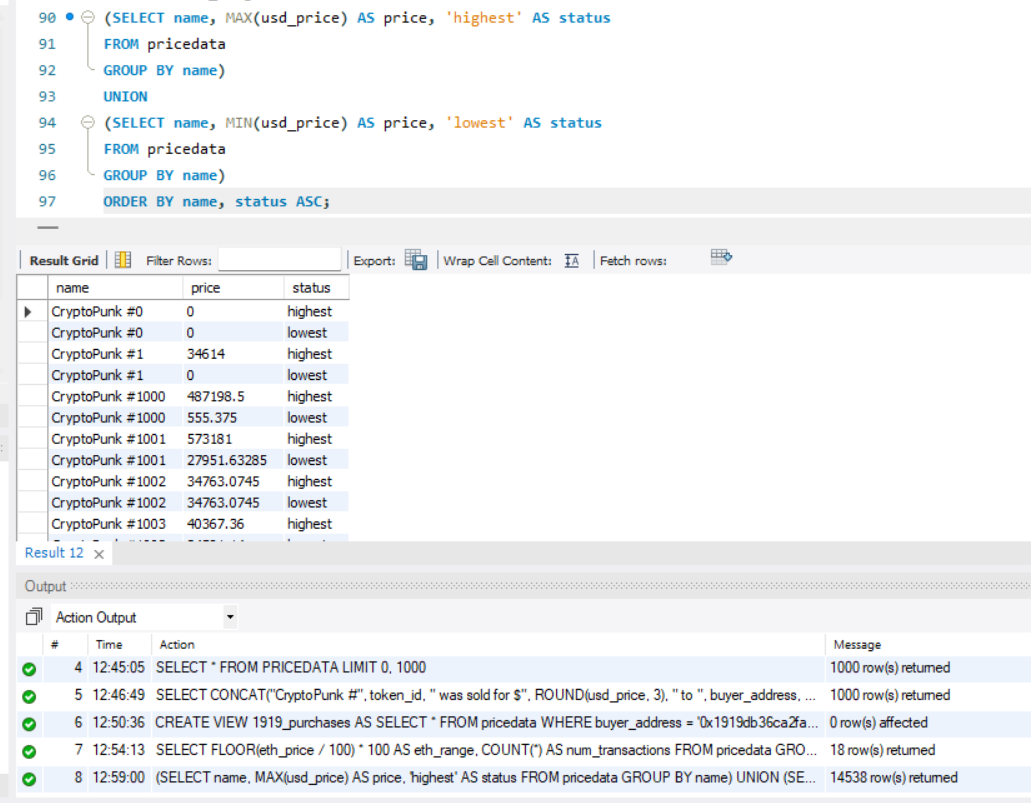
(SELECT name, MIN(usd\_price) AS price, 'lowest' AS status

FROM pricedata

GROUP BY name)

ORDER BY name, status ASC;

```



10. \*\*What NFT sold the most each month/year combination? Also, what was the name and the price in USD? Order in chronological format.\*\*

```sql

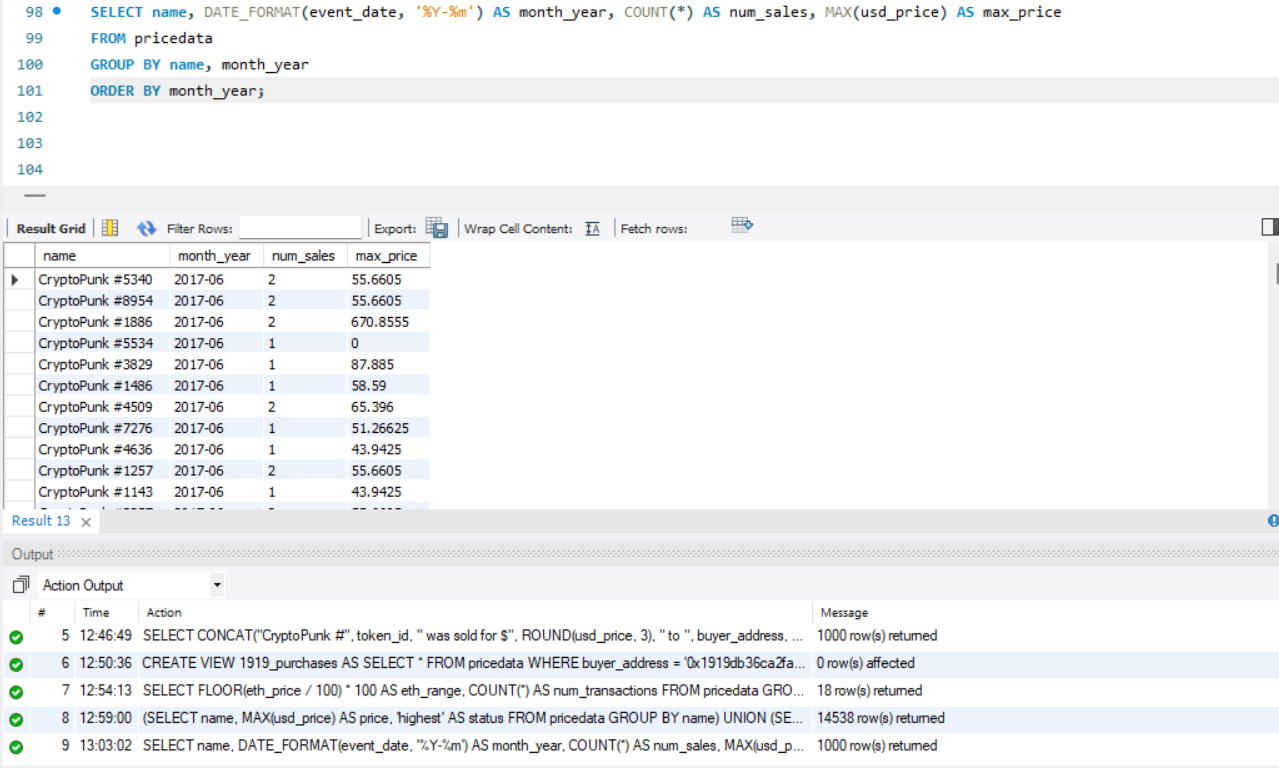
SELECT name, DATE\_FORMAT(event\_date, '%Y-%m') AS month\_year, COUNT(\*) AS num\_sales, MAX(usd\_price) AS max\_price

FROM pricedata

GROUP BY name, month\_year

ORDER BY month\_year;

```



11. \*\*Return the total volume (sum of all sales), round to the nearest hundred on a monthly basis (month/year).\*\*

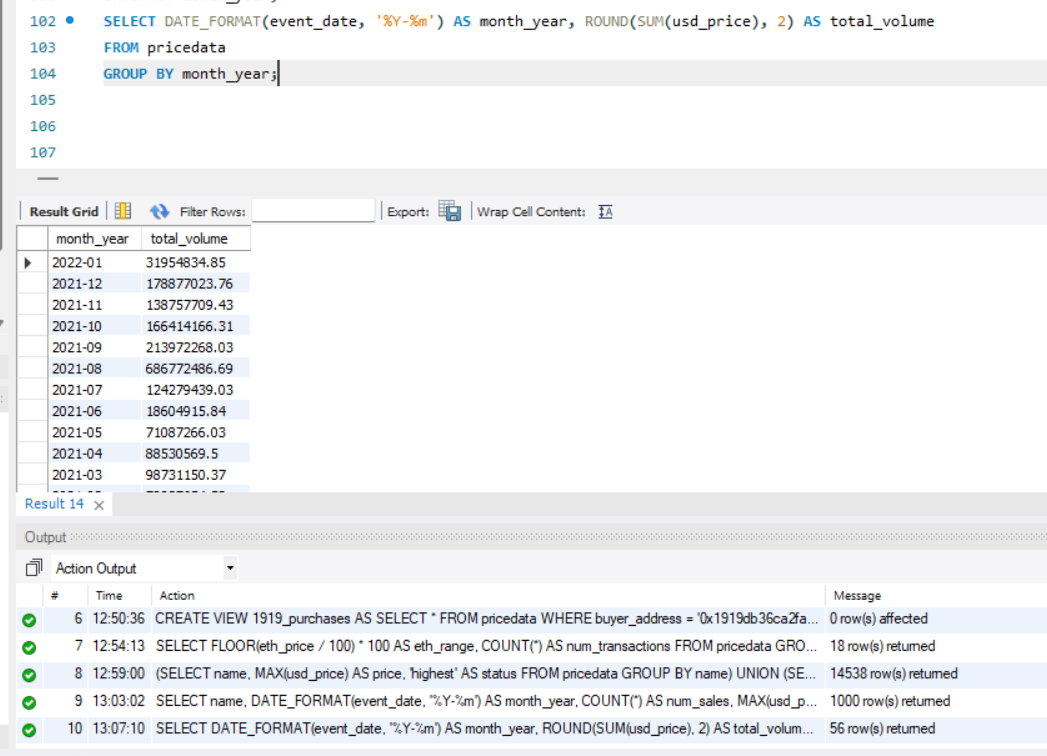
```sql

SELECT DATE\_FORMAT(event\_date, '%Y-%m') AS month\_year, ROUND(SUM(usd\_price), 2) AS total\_volume

FROM pricedata

GROUP BY month\_year;

```



12. \*\*Count how many transactions the wallet "0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685" had over this time period.\*\*

```sql

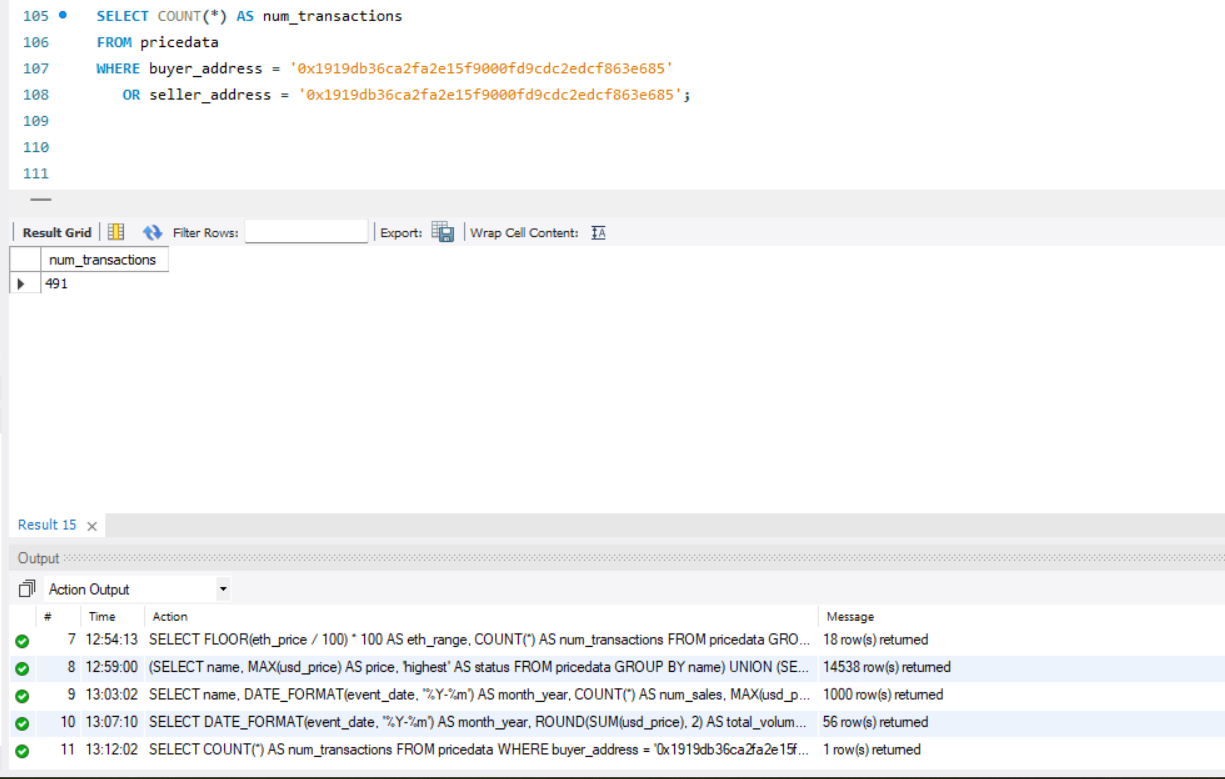
SELECT COUNT(\*) AS num\_transactions

FROM pricedata

WHERE buyer\_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685'

OR seller\_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';

```



13. \*\*Create an “estimated average value calculator” based off of daily transactions.\*\*

```sql

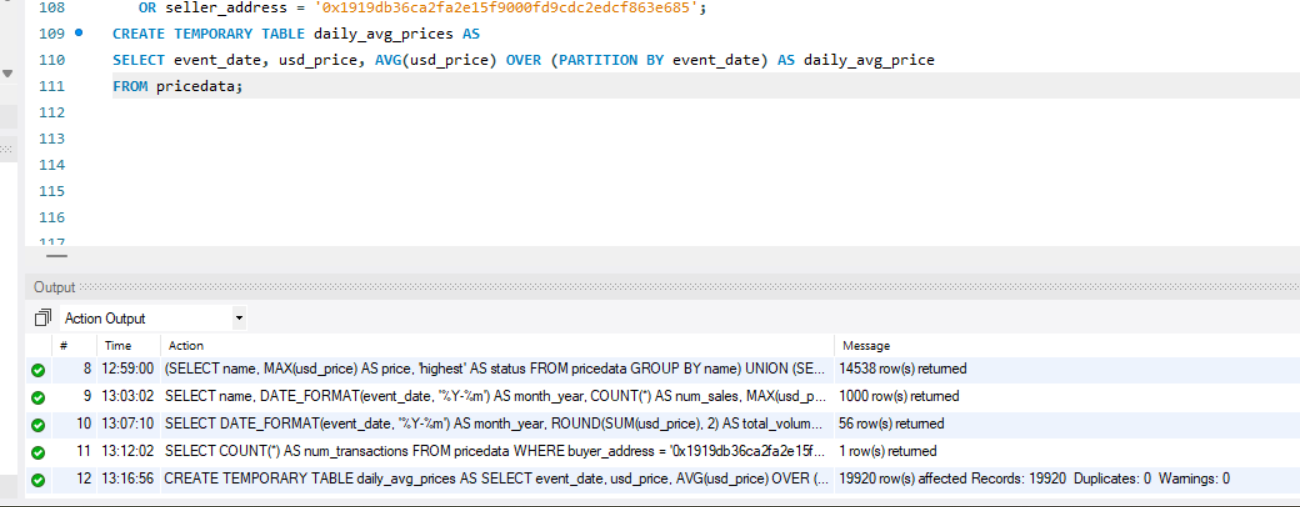
1. First create a query that will be used as a subquery. Select the event date, the USD price, and the average USD price for each day using a window function. Save it as a temporary table.

```sql

CREATE TEMPORARY TABLE daily\_avg\_prices AS

SELECT event\_date, usd\_price, AVG(usd\_price) OVER (PARTITION BY event\_date) AS daily\_avg\_price

FROM pricedata;



1. Use the table you created in Part A to filter out rows where the USD prices is below 10% of the daily average and return a new estimated value which is just the daily average of the filtered data.

```sql

SELECT event\_date, AVG(daily\_avg\_price) AS estimated\_average\_value

FROM daily\_avg\_prices

WHERE usd\_price >= (0.1 \* daily\_avg\_price)

GROUP BY event\_date;

